Pattern Languages in HCI: A critical review

Andy Dearden	Janet Finlay
School of Computing & Management Sciences Sheffield Hallam University Sheffield, S1 1WB, UK Tel: +44 (0)114 225 2916 Fax: +44 (0)114 225 3161 Email:a.m.dearden@shu.ac.uk	School of Computing Leeds Metropolitan University The Grange, Beckett Park Leeds, LS6 3QS, UK Tel: +44 (0)113 283 2600 Fax: +44 (0)113 283 3182 Email: j.finlay@leedsmet.ac.uk

Abstract

This paper presents a critical review of patterns and pattern languages in human-computer interaction (HCI). In recent years, patterns and pattern languages have received considerable attention in HCI for their potential as a means for developing and communicating information and knowledge to support good design. This review examines the background to patterns and pattern languages in HCI, and seeks to locate pattern languages in relation to other approaches to interaction design. The review explores four key issues: what is a pattern? what is a pattern language? how are patterns and pattern languages used? and how are values reflected in the pattern-based approaches to design? Following on from the review, a future research agenda is proposed for patterns and pattern languages in HCI.

1. Introduction

A pattern may be defined as a structured description of an invariant solution to a recurrent problem within a context. A pattern language is a collection of such patterns organised in a meaningful way. In recent years patterns and pattern languages have attracted increasing attention in human computer interaction (HCI) for their potential in recording and communicating design knowledge and supporting the design process. Patterns and pattern languages are now being developed and presented in a wide range of

HCI areas, including: ubiquitous systems (Roth, 2002), web design (van Duyne et al. 2003), safety-critical interactive systems (Hussey, 1999), multimedia exhibits (Borchers, 2001), hypertext and hypermedia (Rossi *et al.* 1997; Nanard *et al.*, 1998), personal digital assistants (Wier & Noble, 2003), socio-technical systems (Thomas, 2003) and games design (Bjork et al., 2003), as well as more general interaction design languages (Tidwell, 1998, 2003; van Welie, 2003; Laakso, 2003).

Initial efforts exploring patterns tended to focus on specific pattern development, leading to repeated debates on correctness and commonality of form and structure, together with a certain amount of "partisanship" regarding particular pattern approaches. Work in software engineering and in interaction design shows a variety of debates about the nature of 'patterns'. Various common elements are generally agreed to be relevant parts of the presentation of patterns, but different authors give significantly different emphases. The result of this is a field that can be daunting to the newcomer, who may find it difficult to disentangle the conceptual characteristics of the approach and therefore its potential contribution to HCI.

In this paper, we present a critical review of research on patterns and pattern languages in interactive systems design, highlighting four key issues within the field. Our aim is to provide an overview of the field, and identify key literature that may be useful and informative to HCI practitioners and researchers. This review also aims to locate patterns in relation to other established and emerging techniques in interactive systems design such as: guidelines and heuristics (Smith & Mosier, 1986; Nielsen, 1994), style-guides (e.g. Microsoft Corporation, 2003; GNOME project, 2003); participatory design (Greenbaum & Kyng, 1991; Schuler & Namioka, 1993), claims analysis (Sutcliffe & Carroll, 1999; Sutcliffe 2000) and design rationale (MacLean et al., 1991).

We begin by outlining the scope of the pattern endeavour that we will consider. We then present a short history of patterns, beginning with Alexander's exposition in architecture, through work in software engineering, to the consideration of patterns in human-computer interaction, in order to place the latter in its historical context. Our review then examines: different interpretations of the concept of pattern; different ideas on the nature of pattern language, different approaches to the use of patterns within the design process,

and different ideas about the role of values in pattern-supported design, before suggesting an agenda for future research.

2. The scope of this review

This review is addressed to practitioners and researchers in HCI. Consequently, the primary focus is on patterns and pattern languages that discuss interaction and interface design issues. There are, however, a large number of patterns from other domains, e.g. software engineering and organisational design, which may have a bearing on interactions between humans and computers. To avoid extending the scope of our review beyond practical limits, we define three broad classes of software-related pattern and pattern language that may be discussed:

General software design patterns –a problem is stated in terms of desirable qualities of the internal structure and behaviour of software, and the solution is stated in terms of suggested code structures. The majority of patterns in Gamma et al. (1995) fall into this category.

Interface software design patterns – a problem is stated in the domain of desirable interaction behaviours, and the solution is stated in terms of suggested code structures. Examples in this category include: patterns for implementing systems that follow a 'tools and materials' metaphor (Riehle & Zűllighoven, 1995); patterns for implementing digital sound synthesis systems (Judkins and Gill 2000); patterns to implement queuing of interaction events (Wake et al. 1996); patterns for e-commerce agent systems (Weiss 2001); and patterns for mobile services (Roth 2002).

Interaction design patterns – a problem is stated in the domain of human interaction issues, and the solution is stated in terms of suggested perceivable interaction behaviour. A good example in this category is Tidwell's (1998) pattern collection including patterns such as 'go back to a safe place' which advocates providing users with a clearly identifiable way of returning a system to a well known state such as the home page of a website. Borchers (2001) includes three distinct pattern languages, the second of which is composed of interaction design patterns and the third of interface software design patterns.

Based on the definitions above, this review is primarily concerned with ‘Interaction Design Patterns’ and, to a lesser extent with ‘Interface Software Design Patterns’. To set the review in context, it is necessary to consider other literature, particularly from Software Engineering and Architecture. However, within such literature, this review will be restricted to general discussions of pattern languages, rather than discussions of the detailed content of the patterns themselves. Due to space constraints the paper does not provide detailed illustratory examples of patterns, for which the reader is referred to published pattern languages and collections (e.g. Alexander, 1977; Gamma et al. 1995; van Duyne et al. 2003) or the available web-based collections of interaction design patterns (e.g. Tidwell, 1999, 2003; van Welie, 2003; Brighton Usability Group, 2003). The early work of Alexander and colleagues (1975, 1977, 1979, 1982, 1985, 1987) in developing pattern languages in architecture will be considered in order to locate HCI patterns within an appropriate historical context.

3. A Short History of Patterns

3.1 Christopher Alexander

Design patterns and pattern languages arose in architecture from the work of Christopher Alexander and his colleagues. Within his profession his proposals have been controversial (Dovey, 1990) but nonetheless they have captured the public imagination with regard to architecture (King, 1993; Gabriel, 1996b) and have been influential in several other domains.

Alexander’s early work, summarised in ‘Notes on the Synthesis of Form’ (Alexander, 1964), proposed a systemic approach to architectural design problems. The approach involves analytic decomposition of the problem into sub-problems, each characterized by a set of competing forces. By resolving the forces in each sub-problem, and synthesizing the individual solutions, the architect generates a solution to the original global problem. Alexander (1964) even considered the possibility of a computational solution to such problems.

During the period from the mid sixties to mid seventies, Alexander became sceptical of his suggestions in 'Notes on the Synthesis of Form'. In the 1970s and early 80s, he and his colleagues set out to define a new understanding and a new approach to architectural design. Grabow (1983), in his biography, describes the changes in Alexander's thinking during this period as a 'paradigm shift'. The new approach, centred on the concept of pattern languages, is described in a series of books, namely: The Timeless Way of Building (Alexander, 1979); A Pattern Language (Alexander *et al*., 1977); The Oregon Experiment (Alexander *et al*. 1975); The Linz Café / Das Kafe Linz (Alexander, 1982); The Production of Houses (Alexander *et al*., 1985) and A New Theory of Urban Design (Alexander *et al*. 1987). The books were published as a series, and are explicitly given volume numbers, which do not correspond with the chronological order of publication. Volume one of the series (*The Timeless Way of Building*) sets out Alexander's view of how patterns and pattern languages evolve, and how they should be utilized in design. Volume two (*A Pattern Language*) offers one instance of a pattern language. The last four volumes of the series each recount a case study in which the pattern based approach to design was applied.

3.2 Pattern Languages in Software Engineering

In the late 1980s and early 1990s, researchers in software engineering were exploring ways to re-use design knowledge. For example, Coplien (1992) investigated idiomatic styles of C++ code, Wirfs-Brock et al. (1991) examined the design of frameworks that supported effective code re-use, Garlan and colleagues investigated the re-use of formal specifications for a family of products (Garlan & Delisle 1990), and generic software architectures that could be refined to specific implementations (Galan & Notkin 1991, Garlan & Shaw 1993). Alexander's concept of 'design patterns' was noticed in the context of this research (Beck & Cunningham, 1987; Coad, 1992; Anderson, 1993; Coad & Mayfield, 1993; Gamma *et al.* 1993; Anderson et al. 1994). The first conference on 'Pattern Languages of Programming' (PLoP) was held in August 1994 (Coplien & Schmidt, 1995). Since then, PLoP conferences have been held annually (Vlissides, Coplien & Kerth, 1996; Martin *et al.*, 1997; Harrison *et al.*, 1999; PLoP, 1998; PLoP, 1999, PLoP, 2000, PLoP, 2001; PLoP 2002; PLoP, 2003). Other conference series

investigating pattern languages in software engineering have also been established, e.g. EuroPLoP in Europe and KoalaPLoP in Australasia. Another important milestone was the publication of Gamma *et al.* (1995), often referred to as the 'Gang of Four' book, which remains one of best selling books in software engineering.

3.3 Patterns in HCI

Early work on patterns in software engineering included solutions for user-interface software design. Thus, Gamma *et al*. (1993, 1995) include patterns such as 'Observer' (an abstraction similar to the 'Model View Controller' architecture) and 'Decorator' (a software design solution used for embellishments such as scrollable panels). The proceedings of the first meeting of PLoP begin with two papers presenting a single interaction design pattern (Adams 1995) and a pattern language with four interaction design patterns to describe a 'tools and materials' metaphor for user interface design, and seven interface software patterns that help implement such interfaces (Riehle & Zűllighoven, 1995).

In the proceedings of the third meeting (Martin *et al.*, 1997), user-interface patterns were recognized as a discrete area of interest and afforded a separate 'part' of the proceedings, despite being represented by a single paper (Bradac & Fletcher, 1997). In the fourth meeting, four papers were grouped in the proceedings as relating to 'Patterns of Human-Computer Interaction' (see Harrison *et al*., 1999). In 1998, (PLoP, 1998) the organisers grouped the papers using section titles taken from '*A Pattern Language*', with the majority of interaction design patterns appearing in the session 'Zen View' (pattern 134 in *A Pattern Language*). Eight of the papers at the 1998 conference include interaction design or interface software design patterns. In 1999 (PLoP, 1999), four papers addressing user-interface issues appear in a group together with two patterns that are primarily concerned with network performance issues. In recent years PLoP has included only a small number of examples of interaction design patterns.

Whilst the number of interaction design and interface software design patterns appearing in PLoP was falling, interest in patterns at meetings of the HCI community was growing. Patterns workshops have become regular events at CHI (Bayle *et al.* 1998, Griffiths *et al.*, 2000, van Welie *et al.* 2002, Fincher *et al*., 2003), as well as being held at a meeting

of the Usability Professionals Association in 1999 (Granlund & Lafreniere, 1999a), and at Interact in 1999 (Griffiths, *et al.* 1999). Panels were held at CHI 2001(Borchers & Thomas, 2001) and at IHM-HCI 2001 (Griffiths & Pemberton, 2001) Papers discussing the use of patterns have been published at a variety of forums including DIS (Erickson, 2000a), ECSCW (Martin *et al.* 2001), PDC (Dearden et al., 2002; Schuler, 2002) and ACM Hypertext conferences (Rossi, 1997; Nanard *et al.* 1998). More recently a number of interaction design pattern languages have been published in book form, including Borchers' triple languages for the development of interactive exhibits (Borchers, 2001), Van Duyne et al.'s Design of Sites language (Van Duyne et al., 2003) and, most recently, Graham's (Graham, 2003) language on web usability. These developments are consistent with the expectations of the participants in the early PLoP meetings. In their introduction to the proceedings of the first PLoP conference, Johnson & Cunningham (1995) state their expectation that "as the PLoP community grows and matures … PLoP will itself splinter along traditional lines of interest" [ibid. p. ix].

The remainder of this paper will consider four of the key issues that arise within patterns research. We begin with the fundamental question of what is a pattern.

4. Issue 1: What is a pattern?

The debate as to what constitutes a pattern has occupied considerable attention in software engineering and HCI. Lea (1994) describes the term pattern as a 'pre-formal construct', noting that Alexander provides no formal definition. Alexander offers many different descriptions of patterns that are taken up by different authors. Coad (1992) emphasises the idea of patterns emerging from repetitions in human behaviour, quoting Alexander's observation that 'every place is given its character by certain patterns of events that keep on happening there' (Alexander, 1979, as quoted by Coad, 1992, p 152). Gabriel (1996b), Denning & Dargan (1996), Cline (1996) Johnson & Cunningham (1995) and Borchers (2001a) also highlight this view. This viewpoint emphasises patterns as recurrent phenomena or structures that must be observed and discovered. The POINTER project (Martin *et al.*, 2001, 2002) captures just such recurrent phenomena, drawing on examples of common interactions derived from ethnographic studies.

An alternative view highlights patterns as artefacts for the explicit representation of design guidance. Gamma *et al.* (1995) quote Alexander 'Each pattern describes a problem … and then describes the core of the solution …' (Alexander et al., 1977, page x, as quoted by Gamma et al. 1995, p 2). Beck et al. (1996) describe patterns as 'a particular prose form' (ibid. p. 103) and Borchers' (2001a) describes patterns as '… above all, a didactic medium for human readers …' (ibid. p. 361). Schmidt et al. (1996) and Astrachan *et al*. (1998) have a similar emphasis.

For Alexander, there is no contradiction between these views. In *The Timeless Way of Building*, Alexander (1979) posits pattern languages as fundamental to the organisation of building, concluding that 'nothing is made without a pattern language in the makers mind; and what that thing becomes, its depth, or its banality, comes also from the pattern language in the builder's mind …' (ibid. p 224). Later, he argues that '… in a period when languages are no longer widely shared, … it becomes necessary to make patterns explicit, … so that they can be shared in a new way – explicitly instead of implicitly – and discussed in public.' (Alexander, 1979, p. 246). His efforts to explicate patterns gives rise to '*A Pattern Language*' (Alexander et al., 1977). Hence, for Alexander, pattern languages are both a theoretical account of the organisation of the built environment, and specific designed artefacts, whose purpose includes re-invigorating public participation in, and discussion of, architectural design.

In software engineering and HCI it is generally agreed that a pattern is a structured description of an invariant solution to a recurrent problem in context, reflecting Alexander's problem oriented approach. However, such an approach is not universal. A distinction can be drawn between design patterns, which centre on a problem and a proven solution, and activity patterns, which simply provide a description of existing patterns of activity (Bayle et al., 1998). For example, the patterns developed in the POINTER project (Martin et al., 2001; Martin et al., 2002), which seek to summarise findings from ethnographic studies, can be seen as 'activity patterns' in Bayle et al.'s terms. Another area of work in software has proposed the idea of 'AntiPatterns' which are examples of poor design practice together with descriptions of how the design could be repaired (Brown et al., 1996). AntiPatterns have not attracted much attention within HCI, although there was some discussion at the CHI 2000 patterns workshop (Griffiths et

al., 2000), in spite of many collections of examples of bad interaction design, with and without repairs. The validity of AntiPatterns in Alexandrian terms can debated, since patterns are, by his definition, concerned with capturing *good* practice. However, their use in software is relatively common and they do occur in interaction design (see for example, Graham, 2003). Within this review, however, we concentrate on the predominant view, i.e. on 'design patterns'.

4.1 Characteristics of Pattern

A number of researchers have discussed what constitutes a design pattern and what distinguishes it from other design advice. Bayle *et al.* (1998) assert that patterns are notable because they are based on examples, facilitate multiple levels of abstraction, bridge the gap between the physical and the social aspects of design and are amenable to piecemeal development. Fincher (1999) also identifies capture of practice and abstraction as important, but adds: organising principle to relate patterns to other patterns in a way that enables design; a value system that is embodied in the patterns; and a particular presentational style.

Perhaps the most comprehensive attempt to characterise patterns arises from the software engineering literature. Winn and Calder (2002) suggest nine essential characteristics of pattern, some of which reflect attributes also identified by previous researchers. They go so far as to assert that anything that fails to exhibit all of these characteristics is not a pattern. Table 1 compares the position of Winn and Calder with that of a selection of authors in HCI who discuss the nature of design patterns. In this table we indicate a direct statement with a bullet and an implicit agreement (for example through the use made of patterns) with a question mark. We have included distinctions made by different authors, even where these are closely related. For example, while there is obviously a relationship between characteristics 4, 7 and 13, they have subtle differences. Characteristic 4 is concerned with the final artefact developed using a pattern; characteristic 7 is concerned with the how the pattern is validated through being used in successful design; and characteristic 13 is concerned with the process of capturing patterns in the first place.

Table 1 illustrates the level of debate on even the fundamental question of what constitutes a pattern. Some of the requirements laid down by Winn and Calder have not been identified as important in HCI, for example the issue of system hotspots. Others, such as levels of abstraction within a pattern, are perhaps so obviously implied by the generic solution and concrete examples, as not to be stated explicitly by any HCI authors. Similarly, many HCI authors imply the focus on design of an artefact through inclusion of notions such as construction and generativity, although they do not mention this explicitly. It is clear, however, that there is a general agreement within HCI that patterns involve the capture of practice, that pattern *languages*, as opposed to single patterns, are important and that patterns involve questions of value.

.

Characteristic	**Commentary**	Winn & Calder2002	Bayle et al. 1998	van Welie et al. 2000	Granlund et al. 2001	Borchers 2000	Finlay et al. 2002	Fincher & Utting 2002	Erickson 2000a	van Duyne et al. 2003	Tidwell 1999
1. A pattern implies an artefact	A pattern should provide a higher-level picture of the shape of the artefact that it describes. The implication is that patterns must support the design of something.	●		?	?	●	?	?	?	●	?
2. A pattern bridges many levels of abstraction	Designers operate at different levels of abstraction (from conceptual maps to code) so a pattern should provide abstraction of practice at different levels.	●						?		?	
3. A pattern includes its rationale	A pattern should include an explanation of why the solution is recommended, and what trade-offs are involved	●	?	●	?	●	?	?	?	●	?
4. A pattern is manifest in a solution	It should be possible to see the pattern that has been used within the finished artefact, since a pattern relates to both design process and structure.	●									
5. A pattern captures system hot spots	By identifying invariants of good design, patterns also highlight design elements that can change, and the relationships between these and the invariant elements.	●									
6. A pattern is part of a language	Patterns are related to other patterns and work together to resolve the complexity of system design problems.	●		?	●	●	●	●	●	●	?
7. A pattern is validated by use	Patterns can only be proved through evidence of their existence in real artefacts and their contribution to design.	●	?	●	?		●		?		●
8. A pattern is grounded in a domain	Patterns relate to specific domains and have no meaning outside those domains.	●	?	?	●	●	?		●	●	
9. A pattern captures a big idea	Patterns should focus on key, difficult problems within a domain.	●						●			
10. Patterns support a 'lingua franca'	Patterns should support discussions with people who are not specialists in the domain.		●	?	?	●	●	●	●	?	●
11. Different patterns deal with problems at different 'scales'	Some patterns in HCI deal with high-level issues such as business process or task structure, whilst others address low level details of GUI construction such as the layout of tables		●	●	●	●	?	?	●	●	●
12. Patterns reflect design values	Patterns are not neutral but explicitly reflect design values		●	?		●	●	●	●	?	●
13. Patterns capture design practice	Patterns are derived from actual practice not theoretical or conceptual proposals		●	●	?	?	?	●		●	?

Table 1: A comparison of different perspectives on the *essential* characteristics of patterns

4.2 Identifying Patterns

As we have seen, one of the distinguishing characteristics of patterns is that they are derived from practice rather than theory. In *The Timeless Way of Building,* Alexander (1979) describes a process that begins by finding places that exhibit what he calls 'the quality without a name', and then trying to identify the distinguishing characteristics that account for the success of the selected design solution. He then seeks to identify key 'invariants' that are common to *all* good solutions to that design problem and not present in poor solutions.

In software engineering, it is usually agreed that patterns must be discovered by reference to design solutions, rather than being constructed from first principles. Coad (1992) suggests that "patterns are found by trial end error and by observation" [p.153]. Coad & Mayfield (1992) discuss 'discovering' patterns from experience. Gabriel (1996b) and Meszaros (1996) both use the metaphor of 'mining' patterns from existing designs. The mining metaphor has been used in workshops on patterns in HCI (van Welie *et al.* 2002), and many of the patterns offered by Tidwell (1998), van Welie (2003) and Brighton Usability Group (2003) are clearly based on observations of common design solutions.

Pattern mining starts with identification of good practice. However, it is not enough simply to capture good HCI practice: pattern mining requires capture of practice that is both good and significant (Fincher and Utting, 2002). Patterns are not intended to state obvious solutions to trivial problems or to cover every possible design decision, but to capture "big ideas" (Winn and Calder, 2002). A pattern should capture insights about the design that can inform even an experienced designer; explaining not only *how* a problem can be solved but also *why* a design choice is appropriate to a particular context. Fincher (2000) reflects that identifying patterns in HCI, i.e. attributing positive qualities of an artefact to particular facets of the design, may be complicated by the high levels of complexity and context dependence in interaction. For example, certain designs (and patterns) may be appropriate in one culture, but not in another (Hall *et al,* 2003). Other design elements may be appropriate only in the context of a particular 'genre'. These problems are not unique to HCI, nor are they insurmountable. Alexander and colleagues (Alexander, 1979; King, 1993) suggest that different cultures will develop and extend

their own architectural pattern languages. Hall et al. (2003) have suggested incorporating statements relating to cultural setting within the 'context' of individual patterns. Walldius (2002) shows how patterns can be used to describe particular 'genres' of film. Van Duyne et al. (2003) use the idea of 'site genre' as an organising principle within their web design pattern language.

One element that is perhaps unique to interaction design patterns is the need to include the notion of temporality (Barfield et al., 1994; Borchers, 2001). Unlike architecture, HCI deals with an artefact where time is significant and the context of and solutions to interaction problems are liable to be dynamic rather than static. A pattern must therefore be able to capture this temporal interactive element. The use of alternative media (such as video) has been suggested to illustrate interactive time-based solutions (Borchers, 2000) but the fundamental issue of abstracting true interaction rather than simply snapshots of appearance or behaviour remains.

On the other hand, patterns should also embody a time*less* quality, presenting a solution that is applicable regardless of platform or technology. This is arguably a weakness in many current interaction design patterns, which are strongly based on a particular and current user interface paradigm (graphical user interfaces for example). Bayle et al. (1998) suggest that patterns that address interaction issues at a 'high level' of abstraction may be timeless, but that patterns that are closer to the detail of interaction design perhaps necessarily reflect current paradigms.

The lack of variety of good examples and the immaturity of our design field as compared to architecture may lead to weaker examples being used as the basis of patterns in HCI (Fincher, 2002). Many interaction design 'patterns' can be criticized for identifying *common* rather than necessarily *good* practice. We shall return to the discussion of 'good' practice in section 7 where we discuss the role of values in patterns.

4.3 The presentation of patterns

Fincher (1999) indicates that identifying good practice is the "least part of the achievement" in developing patterns. Bayle et al. (1998) note that it is relatively easy to observe phenomena in the world but much more difficult to use these observations to develop and explicate good patterns. In order to be useful, patterns must present an

abstraction of good practice at a meaningful level of granularity. Formulations that are too abstract will be impractical in real design use; those that are too specific will be difficult to re-use in new scenarios. Fincher and Utting (2002) compare abstraction in patterns to good teaching practice: it should facilitate understanding of the principles embodied in specific examples, to identify what is important in the examples. Winn and Calder (2002) suggest that patterns should present knowledge at graduating levels of abstraction.The focus on design patterns as a distinct form for design guidance has led to debates about the content and structure of patterns. In software engineering, a range of alternative formats appear in Beck & Cunningham (1987), Coad (1992), Beck (1994), Beck & Johnson (1994), Gamma *et al.* (1995) and Fowler (1997). Meszaros and Doble (1998) present a pattern language for pattern writing, suggesting a degree of stabilization around certain formats. Sharp *et al.* (2003) report on the way that the format of patterns to support computer science education had to be modified to better suit the needs of their target audience.

In HCI, alternative formats have been followed by Tidwell (1998), Borchers (2001), van Welie *et al.* (2000), Martin et al., (2001), Van Duyne et al. (2003) and Tidwell (2003). Some of these (e.g. Borchers, 2001) reflect the layout and typesetting of *A Pattern Language*, others (e.g. Tidwell, 1998) reflect the style of Gamma *et al.* (1995), still others represent departures from previous forms (e.g. Tidwell, 2003; Martin et al., 2001; van Duyne *et al*., 2003). Representative examples of interaction design pattern forms have been collected in the Pattern Gallery (Fincher, 2000b). Several attempts have been made to identify common elements and to formalise these in some way, for example Griffiths *et al.* (1999) and the pattern language markup language PLML developed at the CHI'2003 workshop (Fincher, 2003). Dearden *et al.* (2002) and Finlay et al. (2002) highlight the degree to which different formats, including abbreviated patterns, affect the use of patterns in practical design settings.

4.4 Patterns, Guidelines and Claims

Advocates of patterns in HCI have often sought to demonstrate clear distinctions between patterns and other forms of design guidance. For example, Borchers (2001) suggests that patterns improve upon style guides, guidelines and standards:

> '… through their structured inclusion of existing examples and insightful explanation not only of the solution, but also of the problem context in which this solution can be used, and the structured way in which patterns are integrated into the hierarchy of the language …' (ibid. p60).

Patterns should also be compared to other efforts to re-use design knowledge such as 'claims' (Sutcliffe & Carroll, 1999; Sutcliffe, 2001). To examine such arguments, we need to clarify both the forms of design guidance being discussed, and the contrasts identified. The following common types of design guidance can be distinguished:

1. style guides, which are specific to an environment or product grouping (e.g. GNOME project, 2003; Microsoft Corporation, 2003);
2. general guidelines applicable to a range of systems (e.g. Smith & Mosier, 1986);
3. standards, which may resemble guidelines, but carry some formal authority (e.g. ISO 9241).s
4. claims, which incorporate both theoretical argumentation and specific illustrative examples (e.g. Sutcliffe, 2001) and
5. heuristics, which are general statements of desirable properties (e.g. Nielsen, 1994).

A number of different aspects of patterns and pattern languages are suggested as distinctive. The major contrasts noted by van Welie (2000), Borchers (2001), Fincher (2000a), and Brighton Usability Group (2003) are:

1. the level of abstraction at which guidance is offered;
2. the grounding of patterns in existing design examples, or 'capture of practice';
3. the statement of the problem addressed by a pattern;
4. the discussion of the context in which a pattern should be applied;
5. the provision of a supporting rationale for the pattern;
6. the organisation of patterns into pattern languages; and
7. the embedding of ethics or values in the selection and organisation of patterns.

To simplify discussion we note that standards are not a distinct form of guidance, but are distinguished by their authority. Indeed, the most commonly used standard in HCI (ISO 9241) includes many sections presented as guidelines (referred to as 'principles' or 'recommendations' within the standard). This leaves four distinct forms of guidance. Hence, we can identify twenty-eight (4 x 7) distinct assertions. For example 'interaction

design patterns differ from heuristics because patterns are grounded in concrete examples'. Examining these assertions it is clear that patterns differ from both style guides (because patterns aim to generalise away from particular implementation environments and from fine detail of user-interface rendering, and patterns discuss the context in which they are applicable), and from heuristics (because patterns identify particular solutions, the context of application, and are supported by a rationale). However, it is more difficult to distinguish patterns from guidelines (e.g. Smith & Mosier, 1986; ISO 9241) and claims.

The following similarities and contrasts can be identified:

1. Patterns, guidelines and claims can all be stated at various levels of abstraction. Some patterns tackle issues at a similar level of detail to typical examples of guidelines, e.g. The Shield (van Welie *et al*., 2000) is comparable with ISO 9241-10 principle 3.3. However, the organisation of guidelines around particular styles of interaction (e.g. 'data entry', 'form filling' or 'menu selection') may lead towards guidelines dealing with fine details of interaction, e.g. the arrangement of options within menus. In contrast, interaction design patterns can address larger scale issues over extended interactions. For examples, see 'Step-by-Step Instructions' (Tidwell, 1998), 'Easy Handover' (Borchers, 2001), or 'Recommendation Community' (van Duyne et al., 2003). Claims can also describe such larger scale design issues.
2. Patterns, guidelines and claims all include examples, but whereas examples in guidelines are usually phrased in general terms, e.g. 'imagine an application that …' (Smith & Mosier, 1986), patterns and claims refer to specific implemented systems. There is a slight difference between patterns and claims in the use of examples. Patterns emphasise their grounding in multiple examples of successful designs, whereas claims emphasise grounding in theory. A theory 'motivates' a claim (Sutcliffe & Carroll, 1998), and the claim 'explains' the design of a single artefact. Sutcliffe (2000) suggests that a pattern may be a 'generic design for' a claim (p. 205).
3. Neither guidelines nor claims include a specific *problem* that they attempt to address.
4. Some guidelines include 'exceptions' to identify situations where they should not be applied, but this is not required in all cases. Claims include a specific scenario in which a particular artefact is used, which indicates a 'context' in which the claim

appears valid. In contrast, patterns aim to characterise a set of possible contexts in which the particular design advice should be followed. Hence the ‘context’ in a pattern may generalize over the ‘context’ for individual claims.

5. Guidelines, claims and patterns all provide some supporting rationale based in both primary research and other literature. The presentation of that rationale is more concise in Smith & Mosier’s guidelines than is the case with typical patterns (e.g. Borchers, 2001; van Welie *et al*., 2000). ISO 9241 does not include the references to the literature within the individual guidelines, instead providing a general bibliography.
6. Cross-referencing is common to guidelines, claims and patterns. However, whilst guidelines include occasional cross-referencing, both patterns and claims emphasise organisation and interdependence. We return to this issue in the next section.
7. At one level, guidelines, claims and patterns all embody design values. However in guidelines and claims these values are implicit, patterns aim to make these explicit (Bayle, 1998), both in the expression of individual patterns and in the way that values inform pattern mining (Fincher and Utting, 2002).

In summary, patterns are potentially more general than existing examples of guidelines, use more specific examples, include the statement of a ‘problem’ that they address, deliberately scope their context of application, and explicitly reflect particular design values. Patterns can be distinguished from claims by the inclusion of a problem statement, the requirement for multiple examples, the treatment of context, and the recognition that a pattern explicitly reflects selected design values. This comparison suggests that claims analysis might be a fruitful approach to the identification of patterns, but there may be a tension between the ‘theoretical and empirical’ grounding of claims, and the ‘value led’ approach of patterns.

5. Issue 2: What is a Pattern Language?

Alexander’s original work was not merely about individual patterns, but was explicitly concerned with the concept of pattern languages. Taken in isolation, patterns are, at best, “unrelated good ideas” (Alexander, 1996). However combined in a language, patterns

provide coherent support for design generation. In this section we examine what this means.

5.1 Pattern languages and pattern catalogues

There are two forms of organisation readily evident in *A Pattern Language*. On the one hand, the patterns are collected into sets according to levels of physical scale, e.g. the first section of the language addresses the size and distribution of towns and cities, whilst later sections address smaller units such as neighbourhoods, clusters of houses and individual rooms. In addition, the patterns form a network, where each pattern contains backward references to patterns that set its context, i.e. patterns that have already been used or selected, and forward references to patterns that can be used to help realise the current pattern. For example: the STREET CAFÉ pattern, begins by discussing patterns such as IDENTIFIABLE NEIGHBOURHOOD, ACTIVITY NODES, and SMALL PUBLIC SQUARES that provide contexts to which a street café will contribute and ends by directing the reader to patterns that help realise the street café such as creating an OPENING TO THE STREET, making the terrace double as A PLACE TO WAIT, and using DIFFERENT CHAIRS. This directed network structure provides for Alexander's analogy with the production rules of a grammar (Alexander, 1979, p187).

In contrast, Gamma *et al.* (1995) describe their efforts as a *catalogue* of patterns that have some interrelationships, but do not form a pattern language in Alexander's sense. Gamma *et al*. classify their patterns by their area of concern: creation of objects, structuring of software systems or dynamic behaviour of systems. Other authors who have used classification schemes to organise pattern collections include Kendall *et al*. (1998), Roth (2002), Mahemoff & Johnston (1998) and Hussey and Mahemoff (1999). One of the early OOPSLA workshops in which patterns were a major topic was concerned with creating a 'handbook for software architects' (Anderson,1993). Coplien & Schmidt (1995), discuss the distinction between pattern languages and catalogues, and suggest that

> 'it is likely that catalogs of patterns … will provide the most payoff for pattern based software development over the next few years. It turns out that comprehensive pattern languages … are challenging to produce …' [ibid. p322].

Gamma *et al*. (1995) express the hope that as more patterns are collected their catalogue might evolve and be organised into a language.

Some authors in software engineering, have applied the concepts of refinement and specialisation to examine relationships between patterns. For examples see, Yacoub & Ammar (1998), Mikkonnen (1998), Agerbo & Cornils (1998) and Tahara et al. (1999). A similar approach for interaction design patterns is suggested by Mullet (2002), who proposes three possible relationships between patterns, namely: derivation, where one pattern inherits elements from a higher level pattern; aggregation, where one pattern is contained within another pattern; and association, where one pattern uses another. Van Welie (2003) suggests a similar set of connections between patterns.

A number of pattern collections have been presented using a layered approach, with sets of patterns addressing different 'levels' of a design problem. For example, Tahara *et al*. (1999), provide patterns addressing macro-architectural, micro-architectural, and finally object levels for the design of agent systems. Paternò (2000) suggests 'task patterns' described in the ConcurTaskTrees notation, which are in turn linked to software 'architectural patterns' that are described by configurations of re-usable interaction components called 'interactors'. Granlund *et al.* (2001) suggest interaction design patterns at the levels of 'business domain', 'business process', 'task', 'conceptual design' and 'design'.

5.2 The organisation of pattern languages

Whilst the majority of work in the PLoP conferences has been in the form of individual patterns or pattern collections, a number of networked languages have been presented. For examples, see Richardson (2001), Hanmer (2000), Buschmann (2001) and Dyson & Anderson (1997). Networked pattern languages for interface software include: Riehle & Zűllighoven (1995), Bradac & Fletcher (1997), Towell (1998), Coldewey (1998), Judkins & Gill (2000) Marick (2000) and Berczuk et al. (2000). Richardson (2001) and Hanmer (2000) use an 'enables' relationship between patterns, where later patterns enable the realisation of earlier patterns. Buschmann (2001) selects the term 'completes' to express the relationship between patterns. This relationship in which one pattern 'completes' another at a higher scale is evident in Alexander's writing, particularly in *A New Theory*

of Urban Design (Alexander et al., 1987). Tidwell's (1998) interaction design patterns are networked in a similar way. Borchers (2001) provides three examples of networked pattern languages for: creating blues music, interaction design for multimedia exhibits and interface software design for multimedia exhibits. Van Duyne *et al.* (2003) provide a networked 'language' for the design of websites.

Fincher and others have drawn attention to the issue of the 'organising principle' of pattern languages in HCI (Fincher and Windsor, 2000; Fincher and Utting, 2002; Fincher, 2002). Fincher and Windsor (2000) identify four requirements for an organising principle for a pattern language: it should provide a taxonomy to enable the user to find patterns; it should allow users to find related or proximal patterns; it should allow the user to evaluate the problem from different standpoints; and it should be generative, allowing users to develop new solutions. The two stage organising principle that they propose focuses on the activities of design and the physical characteristics of interface elements rather than the activities of use. This focus is similar to that of other collections such as Tidwell (1998). Van Duyne et al. (2003) group their web-design patterns to address different design aspects, beginning with 'site genre', then examining issues such as 'writing and managing content', and 'making site search fast and relevant'. Van Welie (2003) proposes a layered structure with patterns organised by: posture, akin to Van Duyne et al.'s genres; experience, relating to the particular expectation of the user in approaching the system; task, relating to sequences of interactions; and activity, relating to low level actions. The layers provide a mechanism for grouping the patterns but it is not clear how the relationships between the patterns are determined by it.

These structuring proposals all provide a way of taxonomising a pattern collection, but they do not actively support the process of identifying new patterns. The organisation is not *predictive*. Fincher (2002) contrasts this with other domains, notably chemistry, where the periodic table facilitated the discovery of previously unknown elements, because the organising structure illuminated "gaps" where these could fit. Fincher argues that the organisation of interaction design patterns by physical elements or common uses is arbitrary, whereas Alexander's patterns are organised by the "particular quality of the relationship between physical and psychosocial space" (ibid. p.3). The former could be characterised as a structure; while the latter includes a clear structuring *principle.* Fincher

(2002) suggests that Cognitive Dimensions (Green and Blackwell, 2003) might be a candidate for a structuring principle for interaction design patterns.

5.3 Notions of generativity

A key concept in distinguishing pattern collections from pattern languages is the idea of generativity. Alexander explicitly invokes comparison with generative grammars (see Alexander, 1979, p 187). One reading of the organisation of *A Pattern Language* (Alexander et al., 1977) suggests the idea of generating designs by implicit sequencing of decisions, derived by traversing the network of links between the individual patterns. This understanding is consistent with Alexander's description of case-studies in *The Oregon Experiment* and *The Production of Houses* (Alexander et al., 1975, 1985).
In software engineering, a number of authors have sought to emulate this idea of a generative language. Beck & Cunningham's (1987) suggest that a pattern language helps designers to ask and answer the right question at the right time, i.e. the language can be used to sequence design decisions. Beck (1994), Lea (1994) and Tahara *et al.* (2001) also suggest using the language for sequencing. The idea of patterns being connected by an enabling relationship, where later patterns enable the realization of earlier patterns is apparent in pattern languages in both software engineering and HCI. For examples see: Aarsten et al. (1996), Dyson and Anderson (1997). The notion of an 'enables' or 'completes' relationship between patterns (Richardson, 2001; Hanmer, 2000; Buschmann, 2001) is consistent with this reading of 'generative', in the sense that a higher level pattern implies the use of the lower level patterns that enable it. In HCI Borchers' (2001) suggests this notion of generative sequencing of design decisions, which is also adopted by Finlay et al. (2002). Fincher and Windsor (2000) also reflect this by incorporating design process into their organising structure for pattern languages. However, this is not the only way that the term 'generative' has been discussed in software engineering and HCI. Gabriel (1996a) suggests that individual patterns can be considered 'generative' because they give indirect advice about what to do to achieve a desirable outcome, rather than simply stating that the outcome is desirable. He gives the example of telling himself to 'follow through' when hitting a tennis ball. Lea (1994) also emphasises this notion of generativity, as do Mahemoff & Johnston (1998).

Beck & Johnson (1994) suggest using patterns to construct a more complete design rationale for a whole system, analogous to a mathematical proof. In this analogy, patterns correspond to axioms (or theorems) of the design space. This approach is similar to Thimbleby's (1990) concept of 'Generative Usability Engineering Principles', which specify constraints on permissible designs to ensure that resulting designs exhibit desirable properties. This may also be consistent with Alexander's analogy between pattern languages and Chomsky's grammars and with Alexander et al.'s (1987) approach in '*A New Theory of Urban Design*', and in '*Notes on the Synthesis of Form*' (Alexander, 1964), both of which can be interpreted as forms of design by constraint solving.
Another concept of 'generative' discussed in HCI, is the idea of generating an option space of alternative designs from which the design team should select (Lane, 1990; MacLean et al., 1991; Dearden & Harrison, 1997). Some pattern collections offer the reader a choice of alternative (incompatible) solutions to a design problem, from which one must be selected, based on specified attributes of the domain. For examples in software engineering see McKenney (1996), Dyson and Anderson (1997), Sandu (2001), Tahara et al. (1999, 2001) Mai & de Champlain (2001), Souza *et al.* (2002). In HCI an example is Tidwell's (1998) alternative patterns 'tiled working surface' and 'stacked working surface'.

6. Issue 3: How are patterns and pattern languages used?

Alexander and colleagues provide four books in which they describe various experiments applying pattern based design (*The Oregon Experiment,1975; The Production of Houses,1985; The Linz Café, 1982; and A New Theory of Urban Design, 1987*).
In the field of software engineering, although many patterns, pattern collections and pattern languages have been published, there has been comparatively little discussion of the practical aspects of using patterns. Beck *et al*. (1996) reports on a panel discussion comparing experiences between various software organisations and Fraser *et al*. (1997) debate whether frameworks and patterns actually reduce design costs. We have not found any published details of observational or empirical studies of software developers using patterns in practice.

Similarly, in HCI, there has been relatively little written about the practical details of using patterns in design projects (van Welie *et al.* 2000). Borchers (2002) discusses how patterns might be applied at different stages of Nielsen's (1993) usability engineering lifecycle, and reports that patterns were used by various design teams in developing musical exhibits, but does not discuss precise details of the design activity. Windsor (2000) describes using patterns to capture design rationale within specific projects. The Participatory Patterns Project (Dearden et al. 2002 a, b; Finlay et al. 2002) have reported on simulated design exercises supported by patterns. Borchers (2002) reports on the use of patterns for teaching interaction design. In this section we consider these proposed uses of interaction design patterns in more detail.

6.1 Patterns for participatory design

Alexander argues that user participation in design is essential to successful building: "… it is virtually impossible to get a building that is well adapted to these needs if the people who are the actual users do not design it." (Alexander et al., 1975, p.42). His pattern language was intended to enable users to actively and directly design their own living and working spaces, in part by providing a common language with which they could make proposals and discuss ideas with an 'architect-builder'. A similar emphasis on the need to develop a shared language is apparent in the participatory tradition in HCI (Ehn and Sjőgren, 1991; Ehn and Kyng, 1991; O'Neill, 1998). King (1993) points out that a community using a pattern language in architecture is likely to evolve and develop their own specific pattern language or dialect.

Several authors in HCI have recognised this participatory focus. Bayle, *et al*. (1998) highlight participatory design as one possible application for pattern languages. Borchers (2001) also mentions participatory design as a possibility. The Participatory Patterns Project (Dearden et al. 2002a, b; Finlay et al. 2002) has investigated ways of combining pattern languages with other techniques for participatory interaction design, such as paper prototyping, and has found the approach promising.

A variation on the use of patterns in concert with paper prototyping, is work by Lin and Landay (2002) who propose to integrate patterns into a design sketching environment, allowing designers to drag and drop patterns into their sketches and customise them to

meet local requirements. While this approach is intended for experienced designers, its potential application within participatory design to support early prototyping with patterns is clear.

6.2 Patterns as technical lexicon

Many authors in software engineering suggest using pattern names as a specialist technical lexicon to support design debates. For example, Schmidt (1995) suggests that a knowledge of patterns "helped experts document, discuss and reason systematically about sophisticated architectural concepts" (ibid. p. 70). Cline (1996) suggests that patterns provide a 'standard vocabulary' amongst developers. Meszaros (in Beck *et al.*, 1996) states a similar view. This standard vocabulary can also benefit design documentation, since a pattern name might be sufficient, in some contexts, to explain a complex design. Du & England (2001) propose augmenting the User Action Notation (Hartson *et al*, 1990) with references to patterns in order to produce more concise design specifications. Cline (1996), Schmidt (1995) and Astrachan & Wallingford (1998) all suggest using patterns to educate novices about good software design, and to integrate novices into design teams. Astrachan *et al.* (1998) claim that patterns should form an essential part of the undergraduate computing science curricula. The explicit presentation of the content of patterns may also ease communication across development teams (Schmidt, 1995, p.69). Goldfedder and Rising (1996) suggest using patterns to inform the review of a design, and to aid documentation.

The use of patterns as an educational tool is carried through in HCI. One of the earliest HCI publications on patterns focuses on the use of patterns within an interaction design curriculum (Barfield et al., 1994). Borchers (2002) suggests two ways of using patterns within the curriculum: as a tool to present HCI design knowledge to students and as a methodology to support design. His experiences suggest that both can be successful and that students can grasp the patterns concept. Seffah (2003) and Sharp *et al.* (2003) take this a step further by suggesting the use of pedagogical patterns to design courses, as well as teaching interaction design and process patterns.

Cline (1996) advocates these ways of using patterns, but also suggests that patterns can be used pro-actively to suggest design structures. Where this pro-active design generation

is applied, Cline suggests that designers must apply a degree of 'high-level pattern matching' (ibid. p 47) to identify which patterns to use, and concludes that 'the design patterns must be part of one's flesh and blood – looking things up in a book would be completely unacceptable in these on-the-fly situations" (p47). Goldfedder & Rising (1996) and Buschmann et al. (1996, p423 ff) voice a similar concern that the time to find a pattern increases as more and more patterns are published. This situation may suggest that designers will need to search a database of patterns to find one that matches their current problems rather than making 'on-the-fly' connections as Cline suggests.

6.3 Patterns as organisational memory

In both HCI and software engineering, there has been some work on using patterns as part of an organisational memory. Beck et al. (1996) discuss efforts within specific organisations both to use patterns and to develop patterns that are specific to the domains in which those organisations operate. May and Taylor (2003) propose patterns as a tool for organisational knowledge management. In HCI, Henniger (2001) suggests a process where each development project begins by interrogating a corporate memory to retrieve and select patterns (and guidelines) to use within the project. Relevant patterns are identified by a rule-based system that matches patterns and guidelines against project characteristics (such as user populations, tasks and GUI tools). The selected patterns are then passed to the project to consider. At the end of the project any patterns used are reviewed and may be updated based on the experience gained. Granlund et al. (2001) also suggest updating patterns on the basis of project experiences. Alexander *et al*.'s (1975) suggestions for the management of the pattern language in *The Oregon Experiment* (ibid. p136 ff.), part of which is an annual public review of the pattern language, can also be viewed as a form of organisational learning.

This context of organisational memory has led to the development of a number of tools to support the editing of patterns and pattern languages. Borchers (2001, p 195ff.) describes requirements for PET a 'Pattern Editing Tool'. Schuler (2002) and colleagues are developing an on-line pattern submission and discussion environment for recording patterns for 'living communication'. This environment allows participants to submit and edit their own patterns, and allows members of the public to review submissions.

Some authors have investigated incorporating software patterns into development tools, or implementing patterns as components of programming languages (see, e.g. Agerbo & Cornils, 1998; Mapelsden *et al*., 2002; Chambers *et al*., 2000). This has also been proposed in interaction design (Molina, 2003; Lin & Landay, 2003). It can be objected that such efforts only incorporate the 'solution' part of the pattern, but do not provide advice to software designers about when to use that particular pattern.

6.4 Patterns as lingua franca

Crocker (in Beck et al. 1996) and Beck (1994) both discuss using patterns to support communication between designers responsible for the definition of the overall architecture of a system, and designers responsible for applications software. Schmidt (1996) suggests using patterns to explain architectural design issues to managers. Fowler (1997) suggests using his patterns in collaboration with requirements analysts, clients and domain experts to develop specific models for particular projects.

In HCI, Erickson (2000) also suggests patterns as a 'lingua franca' to support and enhance communication about design, in particular advocating the use of patterns to help users to engage with design processes. Granlund *et al.* (2001) suggest a design process of four phases: system definition; user profiling and task analysis; conceptual design; and 'design'. In each phase, patterns are used as archetypes to begin design discussions with users and clients. Borchers (2001) reports on the use of three separate pattern languages, addressing different aspects of the design of a multi-media exhibit, namely: designing and playing a piece of blues music; designing user interaction for the exhibit; and designing software to implement the exhibit's musical synthesis capabilities. Borchers suggests that, because the pattern format is familiar to designers from each of these different disciplines, they can more readily share their design thinking with each other across disciplinary boundaries. Martin et al (2001; 2002) use patterns (although not design patterns) to present findings from ethnographic studies in a form that might be applied by software designers. Fernández *et al*. (2002) express the hope that their patterns for groupware will improve communication within development teams, between development teams and end users, and between end users. Denning & Dargan (1996) express the hope that a pattern language could provide 'a method of mapping from

human actions to software functions in a way that is intelligible to clients, designers and engineers simultaneously' (ibid., p114). In the Participatory Patterns project, patterns are used to facilitate communication between users and website designers (Finlay et al., 2002, Dearden et al., 2002).

6.5 Patterns as design rationale

As we noted in the discussion of 'generativity', there are a variety of understandings about the semantic relationships between patterns, pattern languages and the designs produced from patterns. There is general agreement that patterns provide some rationale for particular design decisions, but the suggested (or implicit) structure of such rationales differs between authors.

Each of Alexander's patterns contains a discussion of the issues that surround the problem that the pattern addresses, and explains why the chosen solution is desirable. Cline (1996) argues that patterns can provide software engineers with design elements that have 'well-understood trade-offs' (ibid. p. 47). Each of Gamma *et al.*'s (1995) patterns includes discussion of the trade-offs involved in selecting and using it. Additionally, within the 'implementation' section of some of Gamma *et al*'s patterns (e.g. FACTORY METHOD, STATE), alternative design options for certain aspects of the pattern are offered together with advice on selection.

Unlike Alexander's original work, some pattern languages in software engineering offer alternative patterns for similar problems, but designed for different contexts (e.g. Adams et al., 1996; Dyson and Anderson, 1997; McKenney , 1996; Sandu, 2001; Mai & de Champlain, 2001). Fowler (1997) prefers to offer multiple ways of addressing a problem within a single pattern. Tahara et al. (2001) define the context in which each of their patterns should be applied using a common set of indexing attributes. Souza *et al.* (2002) take a similar approach. Coplien (1998) uses tables to relate the selection of certain patterns to analyses of commonalities and variabilities within a domain. If a design rationale notation such as Questions, Options and Criteria (QOC) (MacLean *et al.*, 1991) were employed, approaches such as those described above could be interpreted as treating patterns as re-usable components of rationale.

Fowler (1997) suggests that his patterns can be used to suggest options for a design, which may be accepted, modified or rejected. However, when the pattern is modified or rejected, the justification for that decision should be recorded as part of the design rationale. Again, this could be recorded using a notation such as QOC.

Beck & Johnson's (1994) analogy with axiomatic mathematical proof suggests a more complete rationale connecting all of the design decisions. This view requires a pattern language that is 'generative' in the strict sense of a generative grammar, with the rationale for a design corresponding to a parse tree. However, each of their patterns includes a 'pre-conditions' section restricting the scope of the pattern, e.g. "you are writing a program that is animating a visual display in real time, probably in response to user input …" (Beck & Johnson, 1994, p147). Hence the design rationale would be a proof that the pattern language (the set of axioms) entails the proposition that the specified context implies the selection of the chosen design. This view highlights the fact that the context of a pattern is composed of two different parts. On the one hand, there is a context defined by the position of the pattern in the language, i.e. the larger patterns that it enables; on the other hand, part of the context refers to the nature of the environment in which the pattern is to be applied, the pre-conditions.

In HCI, different authors reflect these different understandings of design rationale. Pattern languages that make use of 'enabling' links to generate designs are consistent with Beck & Johnson's (1994) idea of a proof (see Borchers, 2001; Dearden et al., 2002a, b; Finlay et al., 2002; Riehle & Zűllighoven, 1995; Bradac &Fletcher, 1997; Towell, 1998; Coldewey, 1998; Judkins & Gill, 2000; Marick, 2000). However, these examples do not specify additional contextual details for each individual pattern. Rather the designer must make an initial decision about whether the language is relevant and, if so, the validity of the language and its correct application provides the rationale for the generated design. Tidwell (1998) provides a generative language but does include some patterns that represent distinct alternatives for similar problems (e.g. tiled working surface and stacked working surface). However, she does not specify in detail how to select between these options. Van Duyne *et al* (2003) provide some alternative patterns (e.g. 'fixed width screen size' and 'variable width screen size') together with textual

discussion of suitable contexts for the application of each alternative, which would enable a form of rationale closer to Beck & Johnson's (1994) approach.

Pattern collections and catalogues, cf. van Welie (2003), Henniger (2001), suggest a greater emphasis on pattern matching to construct the rationale. Granlund *et al*'s. (2001) approach also emphasises a rationale constructed by comparing pattern contexts with the conditions of a specific project. This approach is similar to Fowler's (1997). Mahemoff and Johnston (1998) and Hussey & Mahemoff (1999) begin with an analysis of relevant usability dimensions, which is similar to Tahara *et al*.'s (1999) approach, but they do not take this further into a defined process for using patterns.

Windsor (2000) reports on the use of patterns as an explicit mechanism for recording and organising the design rationale in an interaction design project.

7. Issue 4: Values and pattern languages

The idea of a ‘design language’ is well established in the sense of a collection of elements used to create a common design style (Rhienfrank & Eveson, 1996). However, Alexander’s work clearly seeks more than just consistency of style. Rather, the patterns were intended to support a humane architecture that resulted in environments that he describes as ‘living’ and ‘nurturing’. In his keynote address to the annual conference on Object Oriented Programming Systems, Languages and Architectures (OOPSLA) in 1996, Alexander (1996) draws attention to the ‘moral component’ as central to his use of pattern languages in architecture.

“In the architectural pattern language there is, at root, behind the whole thing, a constant preoccupation with the question, under what circumstances is the environment good?” (Alexander, 1996).

This leads to our fourth issue, the place of values in pattern languages for HCI. Issues of value are apparent in patterns in a number of different ways, including:

- The key properties that are examined when attempting to identify ‘good’ design from which patterns may be discovered;
- The selection of, and the rationale provided for, individual patterns;
- The processes by which patterns are recorded and developed;
- The way in which patterns are used.

We examine these aspects in detail below.

7.1 The properties examined to identify patterns

Alexander discusses, at length, ‘The Quality without a Name’. He appeals to this ‘quality’ to distinguish spaces and buildings that are ‘living’ from negative or ‘dead’ spaces. His patterns are then selected to enable the design of such ‘living’ spaces. His procedure for identifying spaces with this ‘quality’ is based on personal observation, but he claims that the ‘quality’ is objective and empirical. To support this claim he reports that when people experience spaces that either do or do not have the quality, they exhibit

a high level of agreement about its presence or absence. This might be interpreted as a claim of inter-rater reliability, though Alexander does not quantify the claim or provide any evidence. What is apparent is the holistic nature of the 'quality' that Alexander is seeking. Dovey (1990) describes Alexander's approach as implicitly phenomenological and suggests that:

"The patterns are derived from the lived world (*lebenswelt*) of everyday experience and they gain their power, if at all, not by being proven empirically correct, but by showing us a direct connection between the pattern and our experience of the built environment." (ibid. p4, author's italics).

In Software Engineering, Gabriel (1996a) focuses on the day to day experience of maintaining a software system. Gabriel suggests , 'habitable code' as a possible analogue for Alexander's 'quality'. Gamma et al. (1995) and Cline (1996) emphasise designing software that is easy to re-use, in particular designing systems that are robust to certain types of change that may be necessary as requirements evolve. Winn & Calder (2002) describe this as identifying system 'hot-spots', i.e. distinguishing aspects of the system that should remain invariant from those that should permit change. Others highlight clarity of communication within development teams and between software development teams and maintenance teams (e.g. see Schmidt, 1995; Cline, 1996; or Beck *et al.* 1996). Both Beck and Meszaros, in their contributions to Beck et al. (1996), describe an aim of saving time in designing software, though Meszaros qualifies this by suggesting that patterns help 'less experienced developers produce good designs faster' [ibid. p112]. Tidwell (1999) criticises software engineers for concentrating on such 'technical' values, and for failing to apply values relating to users' experience of software.

The importance of values was recognised early in the development of patterns in HCI, for example Bayle et al. (1998) discuss this issue. Some authors have sought to identify an analogy for the 'quality without a name' in HCI. Borchers (2001, p. 36) suggests "transparency"; Pemberton posits "engaging" (Pemberton, 2000); Van Welie *et al.* (2000) suggest that 'usability' is sufficient; Finlay et al (2002) compare the 'quality without a name' to Maslow's notion of "wholeness" (Maslow, 1970), which incorporates a sense of unity and integration as an essential component of self-actualisation.

It is not surprising that it is difficult to agree an appropriate analogy for the 'quality without a name' given the holistic and experiential character of the 'quality' described by Alexander. Fincher and Utting (2002) insist that patterns and pattern languages must embody values since they claim to represent 'good' design. Hence, the development of pattern languages challenges practitioners and researchers in HCI to examine the value systems that they employ.

7.2 Values in the selection of and rationale for individual patterns

As well as informing the process of selecting 'good' designs from which patterns might be identified, the individual patterns that are selected and the rationales provided within individual patterns helps to make the authors' design values explicit. For example, Alexander includes patterns such as OLD PEOPLE EVERYWHERE (40) and FOUR STORY LIMIT (21) that clearly reflect particular design values of integrated communities in touch with their environment. In HCI, patterns also reflect the values and priorities of their authors. For example, Borchers's patterns `Attract-Engage-Deliver' and 'Easy Handover' both reflect the value of efficiency, in terms of the flow of people through the exhibition. In the case of the former this is from the perspective of the exhibition sponsor or organiser, wishing to maximise the number of people able to receive the message they wish to deliver. The latter is also concerned with efficiency but has a slightly different focus, reflecting the needs of the user within this rapid turnover. Van Duyne et al. (2003) include a group of six patterns for 'Building Trust and Credibility'. These patterns focus on how designers can create web designs to engender a sense of trust. However, the priority in these patterns is on establishing credibility through external appearance and explicit statements of trustworthiness rather than any attempt to address the actual behaviour (trustworthy or otherwise) of the organisation behind the site.

Values within pattern selection and rationale are reflected in the presentation of patterns at different levels, which provide a value-based context even where patterns cannot be used directly. Alexander includes patterns at a range of levels, from regional and whole city development, through local town planning, to individual neighbourhoods and buildings to interior designs. Clearly not every potential user of the pattern language can exploit all of these patterns: home owners may only be able to use interior design patterns

and some limited architectural patterns, whereas architects, builders and town planners could utilise building and neighbourhood patterns directly.. Relatively few stakeholders are in a position to use the highest level patterns (such as pattern 1, INDEPENDENT REGIONS) directly (although Alexander would argue that each development contributes piecemeal to these global patterns). However, these patterns are important in that they express the values that underpin the authors' view of architectural development, providing context for the lower level patterns. In HCI, there has been little work as yet on such high level, contextual patterns. Perhaps the most relevant work is the Public Sphere Project sponsored by Computer Professionals for Social Responsibility (CPSR) (Schuler, 2002). However it is easy to see parallels in terms of the types of environments, philosophies and scales of development that many researchers and practitioners would wish to promote within interaction design.

7.3 Values in the process of developing patterns

In *The Timeless Way of Building* Alexander (1979) describes the evolution of pattern languages as a social process that is critically dependent on the involvement of users in using and discussing the language and the buildings generated by it (ibid., ch 13). In particular Alexander suggests that professionalisation of debate about design leads people "lose confidence in their own judgement" (ibid. p233) about what designs work for them. From *The Oregon Experiment*, it is apparent that Alexander et al. (1975) expect that specific communities will both adapt existing patterns to suit their needs and will create patterns and pattern languages that are specific to their situation. King (1993) also discusses the development of specific languages within specific communities. This view of the evolution of a pattern language as a social process might be compared with the concepts such as a speech community (Wynn & Novick, 1995), or a genre ecology (Erickson, 2000b).

In software engineering, a specific practice of 'writers workshops', and 'shepherding' has evolved to support the development of patterns and pattern languages. Each workshop has a 'shepherd', who acts as chair and facilitator of the workshop and works with the authors of the papers to initially prepare the paper for the workshop. In the workshop, the

paper is discussed by the workshop participants, but the author(s) are not allowed to comment. Their role is to listen to the discussion. After the workshop, the author(s) take the comments of the workshop into account in finalizing the paper for publication (Kafura et al., 1996; Buschmann et al., 1996; Coplien, 2001). A key value in this process is to ensure that the comments are always constructive, with the appointed 'shepherd' taking responsibility for maintaining the constructive atmosphere.

There is limited evidence of similar pattern writing workshops in HCI. Whilst writers' workshops emphasise pattern writing as a professional albeit apprenticed activity, the Participatory Patterns Project (Dearden et al., 2002; Finlay et al., 2002) reports that use of patterns in a participatory context permitted users to critique and make proposals for change in patterns. This suggests that Alexander's vision of users owning and evolving their own languages may be facilitated by participatory practices.

Another issue that has impacted the development of pattern languages, particularly in HCI, is the distribution of researchers interested in the subject, and the demands on researchers to publish and own work. Bayle et al. (1998) recognise that pattern language development needs to be a community effort, yet the competitive pressures within the wider research context can mediate against such a cooperative approach. This has led instead to competing voices and individual (and often repeated) efforts. Recent moves in developing a shared XML schema for patterns (Fincher, 2003) and the availability of web-based communication systems to permit on-line collaboration in the effort of documenting and distributing pattern languages (for example, van Welie, 2002) are perhaps a move towards a more coherent sharing of the pattern development effort. Schuler (2002) and colleagues are developing an on-line pattern submission and discussion environment for recording patterns for 'living communication'. This environment allows participants to submit and edit their own patterns, and allows members of the public to view currently submitted patterns. It is hoped that this environment will in future support a collaborative process whereby participants can select and develop the patterns towards a coherent pattern language.

7.4 Values in the process of using patterns

Alexander's use of patterns to support participatory design is driven (in part) by a value system that treats localised control, and contextual sensitivity in design as essential. *The Linz Café* (Alexander, 1982) and *A New Theory of Urban Design* (Alexander et al. 1987) discuss the importance of making decisions on the actual construction site, and taking into account the surrounding context. In *The Oregon Experiment* and *The Production of Houses* Alexander et al. (1975, 1985) emphasise the use of patterns by a community to design for itself. In this situation it is important that the written patterns are not regarded as blueprints for design, rather they provide guidance which must be locally interpreted, and must be open to challenge.

In the participatory tradition in HCI there is a similar commitment to users as active participants, rather than passive 'subjects', and to the importance of local context in systems design. As we discussed in section 6.1, The Participatory Patterns Project (Dearden et al, 2002a, b; Finlay et al. 2002) have conducted some initial investigations into this area. However initial results suggest that users may ascribe unwarranted authority to advice presented in the form of patterns (Dearden *et al*., 2002b). To avoid this, the authors advocate encouraging ownership and development of the language by users.

8. Conclusion: A research agenda for patterns in HCI

In this paper we have examined the patterns endeavour in HCI, looking in particular at the nature of patterns and pattern languages, the use that can be made of patterns, and the values they embody. From our review, it is clear that significant contributions have been made in the development of patterns and pattern languages which have been employed in the design of real systems (e.g. Borchers, 2001; van Duyne et al. 2003). However, although the use of patterns is reported, there is little concrete evaluation of either the usefulness of pattern languages within the process or the contribution that they have made to the quality of the end product. Further, discussions of patterns and pattern languages so far within HCI have been dominated by form and examples, with limited examination of the philosophy and values of pattern based design.
Given this situation, we propose that future research should prioritise the following areas.

Enriching pattern languages. The majority of effort to date has been on individual patterns and pattern collections including a considerable amount of duplication. This work needs to be consolidated. We need to develop generative frameworks for organising pattern languages and to focus on patterns at different levels: from the social context of systems to the detail of interfaces. There is also a requirement for a greater focus on timeless patterns as well as those describing a particular platform or technology.

Understanding pattern development. To date pattern development has been relatively ad hoc, based on designer experience and largely individual or small group efforts. This process needs to be better understood and, as Bayle *et al*. suggested in 1998, become a genuine community effort. Pattern mining depends fundamentally on identifying successful design, a process that we need to refine. Frameworks for analysing design to identify the elements that make it successful are needed. The results need to be managed to enable discussion and sharing**.** Languages should be owned by stakeholders and subject to revision, reuse and evolution.

Using patterns in design. One of the most obvious weaknesses in HCI research on patterns to date is the lack of genuine evidence of their benefits to design practice. Perhaps understandably attention has focused on generating patterns, rather than on using them, and most researchers have developed their own languages for a variety of reasons. Significant effort is now required to examine the use of these languages in actual design (e.g. via empirical and observational studies) to demonstrate what benefits might be gained from a patterns approach. Similarly, use may provide one means of validating patterns and languages – those that are useful will (hopefully) develop and grow, those that are not should disappear.

Values in pattern led design. Values is an area where more attention is needed in HCI generally. What values should HCI practitioners and researchers be promoting? What is the equivalent of 'quality without a name' in HCI? How can we identify patterns that are both timeless and culturally sensitive? Understanding the role of values in design may help us to recognise the values embodied in patterns. There are also value issues involved in the development and use of patterns where the need for recognition of contribution needs to be balanced with openness for use and further development. The patterns community may be able to learn here from practices relating to open source software.

Patterns and pattern languages offer an approach to design with much potential. Research in these areas is now needed to ensure that this promise is fulfilled and that pattern language research makes an effective and lasting contribution to the practice and understanding of interaction design.

References

Aarsten,A., Brugali, D. and Menga, G. 1996. Designing Concurrent and Distributed Control Systems. *Communications of the ACM*, 39(10), 50 – 58.

Adams, S., 1995. Functionality Ala Carte. In Coplien & Schmidt, 1995, 1 – 8.

Adams, M., Coplien, J., Gamoke, R., Hanmer, R., Keeve, F. & Nicodemus, K., 1996. Fault Tolerant Telecommunication System Patterns. In Vlissides et al. (Eds.), 1996, 549 – 561.

Agerbo, E. & Cornils, A., 1998. How to preserve the benefits of design patterns. In Proceedings of OOPSLA'98, *ACM SIGPLAN Notices* 33(10), 134 – 143.

Alexander, C., 1964. *Notes on the Synthesis of Form*. Harvard University Press

Alexander, C. 1982. *The Linz Café / Das Kafe Linz.* Oxford, UK. Oxford University Press.

Alexander, C. 1996. *The Origins of Pattern Theory, the Future of the Theory, And The Generation of a Living World*. Keynote address to the eleventh annual conference on Object-oriented programming systems, languages, and applications. October 6 - 10, 1996, San Jose, CA USA. Available from http://www.patternlanguage.com/archive/ieee/ieee.htm

Alexander, C., 1979. *The Timeless Way of Building*. Oxford, UK. Oxford University Press.

Alexander, C., Davis, H., Martinez, J. & Corner, D., 1985. *The Production of Houses*. Oxford, UK: Oxford University Press.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S., 1977. *A Pattern Language.* Oxford, UK. Oxford University Press.

Alexander, C., Neis, H., Anninou, A. & King, I., 1987. *A New Theory of Urban Design*. Oxford, UK. Oxford University Press.

Alexander, C., Silverstein, M., Angel, S., Ishikawa, S & Abrams, D., 1975. *The Oregon Experiment*. Oxford, UK. Oxford University Press.

Anderson, B., 1993. Addendum to the Proceedings of OOPSLA ’92. Workshop Report: Towards and Architecture Handbook. *OOPS Messenger* 4(2), pp. 109 – 113.

Anderson, B., Coad, P and Mayfield, M. 1994. Addendum to the Proceedings of OOPSLA ’93. Workshop Report: Patterns: Building Blocks for Object Oriented Architectures. *OOPS Messenger* 5(2), pp. 107 – 109.

Astrachan, O. & Wallingford E. 1998. *Loop Patterns*. Paper presented at PLoP ’98. Available from: http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/P60.pdf

Astrachan, O., Berry, G., Cox, L. & Mitchener, G., 1998. Design Patterns: An Essential Component of CS Curricula. *ACM SIGSCE Bulletin*, 30 (1), pp. 153 – 160.

Barfield, L., Van Burgsteden, W., Lanfermeijer, R., Mulder, B., Ossewold, J., Rijken, D. and Wegner, P. 1994. Education: Interaction Design. In ACM SIGCHI Bulletin, Vol 26, 3.

Bayle, E., Bellamy, R., Casaday, G., Erickson, T., Fincher, S., Grinter, B., Gross, B., Lehder, D., Marmolin, H., Moore, B., Potts, C., Skousen, G. and Thomas, J. (1998) Putting it all together: Towards a pattern language for interaction. *SIGCHI Bulletin*, 30, 1, 17-33.

Beck, K. & Cunningham, W., 1987. *Using Pattern Languages for Object-Oriented Programs*. Technical Report No. CR-87-43. Tektronix Inc., Available from: http://c2.com/doc/oopsla87.html

Beck, K. & Johnson, R., 1994. Patterns Generate Architectures. In, *Proceedings of the OO Programming 8th European conference (ECOOP 94)* Berlin: Springer. 139-149

Beck, K., 1994. Patterns and Software Development. *Dr Dobbs Journal*, **19** (2). pp. 18 – 23.

Beck, K., Coplien, J. O., Crocker, R., Dominick, L., Meszaros, G., Paulisch, F., Vlissides, J. 1996. Industrial Experience with Design Patterns. *Proceedings of the 18th International Conference on Software Engineering (ICSE 18).* IEEE Computer Society, ISBN 0-8186-7246-3. 103 – 114.

Berczuk, S., Appleton, B. & Cabrera, R., 2000. Getting Ready to Work: Patterns for a Developer's Workspace. Paper presented at PLoP 2000. Available from: http://jerry.cs.uiuc.edu/~plop/plop2k/proceedings/Berczuk/Berczuk.pdf

Bjork, S., Lundren, S. and Holopainen, J 2003 Game Design Patterns Project. Available at http://www.gamedesignpatterns.org/

Blackwell, A. and Green, T. 2003. Notational Systems – The Cognitive Dimensions of Notations Framework. In J. M. Carroll (ed.) *HCI Models, Theories and Frameworks: Toward a Multidisciplinary Science*, Morgan Kaufmann, pp. 103 – 134.

Borchers, J (2000) Interaction Design Patterns: Twelve Theses. Position paper presented at the Workshop on Pattern Languages for Interaction Design, CHI 2000 Conference on Human Factors in Computing Systems, April 2-3rd, The Hague, Netherlands. Available at: http://www.stanford.edu/~borchers/publications/ chi2k/CHI2K-Borchers.pdf

Borchers, J., 2001. *A Pattern Approach to Interaction Design*. John Wiley and Sons, Chichester, UK.

Borchers J., 2002. *Teaching HCI Design Patterns: Experience From Two University Courses*. Position paper for "Patterns in Practice" workshop at CHI 2002, Minneapolis, MI. April 21 – 25, 2002. Available from: http://www.hcipatterns.org.

Borchers, J. O. & Thomas, J. C., 2001. Patterns: what's in it for HCI? Panel Discussion. In, *CHI '01 extended abstracts on Human factors in computer systems*, ACM Press, ISBN 1-58113-340-5, 225 – 226.

Bradac, M. & Fletcher, B. 1997. A Pattern Language for Developing Form Style Windows. In, Martin *et al.*, 1997, 347 – 393.

Brighton Usability Group, 2003. The Brighton Usability Patterns Collection. Available at: http://www.cmis.brighton.ac.uk/research/patterns/home.html

Brown K. and Whitenack, B. *A Pattern Language for Relational Databases and Smalltalk.* Available at http://www.ksc.com/article2.htm

Brown, W. H., Malveau, R. C., McCormick, H. W., Mowbray, T. J., 1998. *AntiPatterns: Refactoring Software, Architectures and Projects in Crisis*. New York, John Wiley.

Buschmann, F. 2001. *A Pattern Language for Distributed Object Computing*. Presented at EuroPLoP 2001. Available at: http://hillside.net/patterns/EuroPLoP2001/papers/Buschmann.zip

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M, 1996. *Pattern-oriented software architecture: a system of patterns*, John Wiley & Sons, Inc., New York, NY, 1996

Buschmann, F., Johnson, R., Coplien, J., Rising, L, Delano, D., Gamma, E. and Schmidt D. 1996 *How to hold a writers' workshop*, written in preparation for PLoP 1996 Available from: http://www.cs.wustl.edu/~schmidt/writersworkshop.html

Chambers, C., Harrison, B. & Vlissides, J. 2000. A Debate on Language and Tool Support for Design Patterns. In *Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM Press. ISBN: 1-58113-125-9, pp. 277 – 289.

Cline, M. P., 1996. The Pros and Cons of Adopting and Applying Design Patterns in the Real World. *Communications of the ACM* **39** (10) 47 – 49.

Coad, P. 1992. Object-Oriented Patterns. *Communications of the ACM*, 35(9), pp. 152 – 159.

Coad, P. & Mayfield, M. 1993. Addendum to the Proceedings of OOPSLA '92. Workshop Report: Patterns. OOPS Messenger 4(2), pp. 93 - 95.

Coldewey J. 1998. User Interface Software. Presented at PLoP '98. Available from: http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/P13.pdf

Coplien, J. O., 1992 *Advanced C++ programming styles and idioms*. Reading MA. USA, Addison-Wesley.

Coplien, J. O., 1998. *Multi-paradigm design for C++*. Addison-Wesley, Reading MA. USA.

Coplien, J. O., 2001. *Writers Workshop Patterns*. Available from: http://c2.com/cgi/wiki?WritersWorkshopPatterns

Coplien, J. & Schmidt, D., 1995. Pattern Languages of Program Design. Reading MA. USA, Addison-Wesley.

Dearden A, Finlay, J., Allgar, E. & McManus, B. 2002a. Evaluating Patterns in Participatory Design. In *Adjunct Proceedings of CHI 2002*, pp. 664 – 665. ACM Press, New York, USA.

Dearden A, Finlay, J., Allgar, E. & McManus, B. 2002b. Using Pattern Languages in Participatory Design. In T. Binder, J. Gregory & I Wagner (Eds.) *Proceedings of PDC 2002*. CPSR: Palo Alto, CA.

Dearden, A. M. & Harrison, M. D., 1997. Abstract Models for HCI. *International Journal of Human-Computer Studies*, 46 (1), 151 - 177.

Denning, P. & Dargan, P., 1996. Action Centred Design. In, Winograd, T. (Ed.) *Bringing Design to Software*. ACM Press. ISBN 0-201-85491-0. pp. 105 – 120.

Dovey, K., 1990. The pattern language and its enemies. *Design Studies* **11**(1), 3 - 9.

Du, M. & England, D., 2001. Temporal Patterns for Complex Interaction Design. In Johnson, C. (Ed.): Interactive Systems: Design, Specification, and Verification, 8th International Workshop, DSV-IS 2001, Lecture Notes in Computer Science 2220, London: Springer. 114-127

Dyson, P. & Anderson, B. *1997.* State Patterns. In Martin *et al*. (Eds.) 1997, 125 – 142.

Ehn, P. & Kyng, M., 1991. Cardboard Computers: Mocking-it-up or Hands-on the Future. In J. Greenbaum and M. Kyng, (Eds.), *Design at Work*. pp. 169-196, Lawrence Erlbaum Associates, Hillsdale, NJ.

Ehn, P. & Sjőgren, D., 1991. From System Descriptions to Scripts for Action, In J. Greenbaum and M. Kyng, (Eds.), *Design at Work*. pp. 241 – 268. Lawrence Erlbaum Associates, Hillsdale, NJ.

Erickson, T., 2000a. Lingua Francas for Design: Sacred Places and Pattern Languages. In *The Proceedings of DIS 2000* (Brooklyn, NY, August 17-19, 2000). New York: ACM Press, 357-368.s

Erickson, T. 2000b. Making Sense of Computer-Mediated Communication (CMC): Conversations as Genres, CMC Systems as Genre Ecologies. In Nunamaker, J. F. Jr. and Sprague, R. H. Jr (Eds.) *Proceedings of the Thirty-Third Hawaii International Conference on Systems Science HICSS-33*. IEEE Press. p3011. A longer version of this

paper is available from: http://www.pliant.org/personal/Tom_Erickson/genreEcologies.html

Fernández, A., Holmer, T., Rubart, J. & Schűmmer, T. 2002. Three Groupware Patterns from the Activity Awareness Family. Presented at EuroPLoP 2002. Available at: http://hillside.net/patterns/EuroPLoP2002/papers/Fernandez_Holmer_Rubart_Schuemmer.zip

Fincher, S. 1999. Analysis of Design: An Exploration of Patterns and Pattern Languages for Pedagogy. *Journal of Computers in Mathematics and Science Teaching: Special Issue on Computer Science Education*, 18 (3), pp. 331-348.

Fincher, S. 2000a. "Capture of Practice": Is it obvious? *BCS HCI Group/IFIP WG 13.2 Workshop on HCI Patterns*, November 2000. Available at http://www.cs.kent.ac.uk/people/staff/saf/patterns/bcs.pdf

Fincher, S. 2000b. The Pattern Gallery. Available at http://www.cs.ukc.ac.uk/people/staff/saf/patterns/gallery.html

Fincher. S. 2002. Patterns for HCI and Cognitive Dimensions: two halves of the same story? In Kuljis, J., Baldwin, L. and Scoble, R. (eds.), *Proceedings of the Fourteenth Annual Workshop of the Psychology of Programming Interest Group*, Brunel University, UK, June 2002, pp. 156-172. Available at: http://www.ppig.org/papers/14th-fincher.pdf

Fincher, S. 2003. PLML: Pattern Language Markup Language. *Interfaces*, 56, British HCI Group. Report of Workshop held at CHI2003, pp. 26-28

Fincher, S. & Utting, I., 2002. Pedagogical patterns, their place in the genre. In proceedings of ITiCSE. June 24th – 26th Aarhus, Denmark. ACM Press.

Fincher, S & Windsor, P. 2000. Why patterns are not enough: some suggestions concerning an organising principle for patterns of UI design, *CHI'2000 Workshop on Pattern Languages for Interaction Design: Building Momentum.* Available at http://www.cs.kent.ac.uk/people/staff/saf/patterns/chi00.pdf

Fincher, S., Finlay, J., Greene, S., Jones, L., Matchen, P., Thomas, J. Molina, P. Perspectives on HCI patterns: concepts and tools, CHI '03 extended abstracts on Human

factors in computer systems, April 05-10, 2003, Ft. Lauderdale, Florida, USA, pp. 1044-1045

Finlay, J., Allgar, E., Dearden, A. & McManus, B., Pattern Languages in Participatory Design, in Faulkner, X., Finlay, J. & Detienne, F. (eds.), *People and Computers XVI - Memorable yet Invisible, Proceedings of HCI2002,* pp. 159-174, Springer Verlag: London.

Fowler, M., 1997. Analysis Patterns: Reusable Object Models. Addison Wesley, Menlo Park, CA., USA. ISBN: 0-20189542-0.

Fraser, S., Beck, K., Booch, G., Johnson, R. & Opdyke, B. Beyond the Hype: Do Patterns and Frameworks Reduce Discovery Costs? *Proceedings of the 1997 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications (OOPSLA '97)* SIGPLAN Notices **32** (10) 342-344.

Gabriel R., 1996a. Introduction to *Pattern Languages of Program Design 2*, Vlissides *et al.* (Eds.) 1996.

Gabriel R. P. 1996b. *Patterns of Software: tales from the software community.* Oxford University Press. ISBN 019510269X.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1993. Design Patterns: Abstraction and Reuse of Object-Oriented Design. In *Proceedings of the 7th European OO Programming conference* ECOOP 93, LNCS 707. Springer, Berlin, Germany. 406 – 431.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J., 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA.

Garlan, D. & Delisle, N., 1990. Formal specifications as reusable frameworks. In Bjorner, B. Hoare, C. A. R. & Langmaack H. (Eds*.) VDM and Z: Formal Methods in Software Development.* LNCS 428, Springer-Verlag, 150 – 163.

Garlan, D. & Notkin, D., 1991. Formalising design spaces: Implicit invocation mechanisms. In, S. Prehn [and] W. J. Toetenel, Eds. *VDM '91: Formal Software Development Methods.*. LNCS 551, Springer-Verlag, 31 – 44.

Garlan, D. & Shaw, M., 1993. An Introduction to Software Architecture. In V. Ambriola and G. Tortora (ed.), *Advances in Software Engineering and Knowledge Engineering,*

Series on Software Engineering and Knowledge Engineering, Vol 2, World Scientific Publishing Company, Singapore, pp. 1-39.

GNOME project, 2003. *The GNOME Human Interface Guidelines.* http://developer.gnome.org/projects/gup/hig/1.0

Goldfedder, B. & Rising, L., 1996. A Training Experience with Patterns. *Communications of the ACM* 39(10) 60 – 64.

Grabow, S., 1983. *Christopher Alexander. The search for a new paradigm in architecture*. Stocksfield, Northumberland, UK: Oriel Press.

Graham, I. 2003. *A Pattern Language for Web Usability*. Addison Wesley: London.

Granlund, A., Lafreniere, D. & Carr, D. A. 2001. PSA: A pattern supported approach to the user interface design process. In *Proceedings of HCI International 2001,* vol. 1, Mahwah, NJ: Lawrence Erbaum Associates pp. 282-286.

A. Granlund and D. Lafreniere, 1999. A pattern-supported approach to the user interface design process. Workshop report, UPA'99 Usability Professionals' Association Conf. (Scottsdale, AZ, June 29-July 2, 1999). Available from: http://www.upassoc.org/conf99reg/ws6.shtml

Greenbaum, J. & Kyng, M. (Eds.), 1991. *Design at Work*. pp. 169-196, Lawrence Erlbaum Associates, Hillsdale, NJ, USA.

Griffiths, R., Pemberton, L., Borchers, J. 1999. Usability Pattern Language: Creating a community. In Brewster, S., Cawsey, A. & Cockton, G. (Eds.) *Human-Computer Interaction – Interact 99 (Volume II).* British Computer Society, ISBN: 1-902505-19-0, p. 135. Outputs from workshop are available at http://www.it.bton.ac.uk/staff/rng/UPLworkshop99/

Griffiths, R., Pemberton, L., Borchers, J., & Stork, A. Pattern languages for interaction design: Building momentum. *CHI2000 Extended Abstracts,* p. 363. ACM Press, 2000.

Griffiths, R. N. & Pemberton, L. (2001). 'Patterns in Human-Computer Interaction Design' (panel session) In, J. Vanderdonckt, A. Blandford, and A. Derycke, (Eds.) *Proceedings of IHM-HCI 2001, volume II,* Toulouse, France, Cépaduès-Éditions.

Hall, P. A. V., Lawson, C. J. & Minocha, S., 2003. Design Patterns as a Guide to the Cultural Localisation of Software. In *Proceedings of the 5th International Workshop on Internationalisation of Products and Systems*, Berlin, 17th – 19th July, 2003, pp. 79 – 88.

Hanmer, R. S., 2000. *Real Time and Resource Overload Language*. Presented at PLoP 2000. Available at: http://jerry.cs.uiuc.edu/~plop/plop2k/proceedings/Hanmer/Hanmer.pdf

Harrison, N, Foote, B. & Rohnert, H. (Eds.) 1999. *Pattern Languages of Program Design 4*. Reading, MA., USA, Addison Wesley.

Hartson, H. R., Siochi, A. C. & Hix, D. (1990). The UAN: A user-oriented representation for direct manipulation. *ACMS Trans. on Information Systems* 8(3): 181 – 203.

Henninger, S., 2001. An Organizational Learning Method for Applying Usability Guidelines and Patterns. In Little, M.R. and Nigay, L. (Eds.) *Engineering Human-Computer Interaction*. LNCS 2254. Springer, Berlin, Germany 141 – 156.

Holtzblatt, K. and Beyer, H. 1997 *Contextual Design*. Morgan Kaufman, San Fransisco, CA, USA.

Hussey, A. 1999. Patterns for safer human-computer interfaces. In M. Felici, K. Kanoun and A. Pasquini, editors, *Computer Safety, Reliability and Security: SAFECOMP'99*, 103-112. Springer-Verlag.

Hussey, A. and Mahemoff, M., 1999. Safety-Critical Usability: Pattern-based Reuse of Successful Design Concepts. In M. McNicol (ed.), *4th Australian Workshop on Industrial Experience with Safety Critical Systems and Software (SCS) 99*, Canberra, Australia, pages 19-34, ACS

International Standards Organisation (n.d.) ISO International Standard 9241 (Ergonomic requirements for office work with visual display terminals) Available from http://www.iso.org/

Johnson, R. & Cunningham, W., 1995. Introduction to Coplien & Schmidt, 1995.

Judkins, T. V. & Gill, C. D. G., 2000. Synthesizer A Pattern Language for Designing Digital Modular Synthesis Software. Paper presented at PLOP 2000, available from http://jerry.cs.uiuc.edu/~plop/plop2k/proceedings/Judkins/Judkins.pdf

Kafura, D., Lavender, G. & Schmidt, D., Workshop on Design Patterns for Concurrent, Parallel and Distributed Object Oriented Systems. In Addendum to the proceedings of OOPSLA '95 *OOPS Messenger* 6(4) 128 – 131.

Kendall E. A., Murali Krishna, P. V., Pathak, C. V. & Suresh, C. B., 1998. Patterns of Intelligent and Mobile Agents. In, *Proceedings of the Second International Conference on Autonomous Agents*. 92 – 99. ACM Press.

King, I., 1993. Christopher Alexander and Contemporary Architecture. Special issue of *Architecture and Urbanism*, August 1993.

Laakso, S. 2003. User Interface Design Patterns, Available at http://www.cs.helsinki.fi/u/salaakso/patterns/

Lane, T. G., 1990. *Studying Software Architecture through Design Spaces and Rules*. Technical Report CMU/ SEI-90-TR-18. Software Engineering Institute, Carnegie Mellon University.

Lea, D. 1994. Christopher Alexander: An Introduction for Object-Oriented Designers, *Software Engineering Notes*, 19 (1) 39-46.

Lin, J. and Landay, J. A. 2002 Damask: A Tool for Early-Stage Design and Prototyping of Multi-Device User Interfaces. In *Proceedings of The 8th International Conference on Distributed Multimedia Systems (2002 International Workshop on Visual Computing)*, San Francisco, CA, September 26-28, 2002, pp. 573-580.

Maclean, A., Young, R. M., Bellotti, V. M. E., & Moran, T. P., 1991. Questions, Options & Criteria: elements of design space analysis. Human-Computer Interaction, **6** (3 & 4) 201 – 250.

Mahemoff, M. J. and Johnston, L. J., 1998. Principles for a Usability-Oriented Pattern Language In Calder, P. and Thomas, B. (Eds.), *OZCHI '98 Proceedings* , Los Alamitos, CA IEEE Press. 132-139.

Mai, Y. & de Champlain, M., 2001. *A Pattern Language to Visitors.* Presented at PLoP 2001. Available from: http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/PLoP2001/ymai0/PLoP2001_ymai0_1.pdf

Mapelsden, D. Hosking, J. & Grundy, J. 2002. Design Pattern Modelling and Instantiation using DPML. In, Noble, J. & Potter, J. (Eds.) Proceedings of the 40th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Pacific 2002), 3 – 11. ACM Press.

Marick, B. 2000. *Using Ring Buffer Logging to Help Find Bugs*. Presented at PLoP 2000. Available from: http://jerry.cs.uiuc.edu/~plop/plop2k/proceedings/Marick/Marick.pdf

Martin, D., Rodden, T., Rouncefield, M., Sommerville, I and Viller, S. (2001) Finding Patterns in the Fieldwork. In Prinz, W., Jarke, M., Rogers, Y, Schmidt, K., Wulf, V. (Eds.): Proceedings of the Seventh European Conference on Computer Supported Cooperative Work, Kluwer 2001 ISBN: 0-7923-7162-3, pp.39-58

Martin, D., Rouncefield, M. & Sommerville, I., 2002. Applying patterns of cooperative interaction to work (re)design: e-government and planning. In *Proceedings of CHI 2002*. ACM Press, 235-242

Martin, R. C., Riehle, D. and Buschmann, F. (Eds.), 1997. Pattern Languages of Program Design 3. Addison Wesley, Reading MA., USA etc. ISBN: 0201310112.

Maslow, A. 1970. *Motivation and Personality, Third Edition*, Harper and Row, London, UK.

May, D. and Taylor, P. 2003. Knowledge Management with Patterns, *Communications of the ACM*, 46(7) 94-99

McKenney, P. E., 1996. Selecting Locking Primitives for Parallel Programming. Communications of the ACM, **39** (10), October 1996, pp. 75 - 82.

Meijler, T., D., Demeyer, S. & Engel, R., 1997. Making Design Patterns Explicit in FACE: a Framework Adaptive Composition Environment. In, *Proceedings of the 6th European conference held jointly with the 5th ACM SIGSOFT symposium on Software engineering*. Springer-Verlag, New York, NY., USA. ISBN: 3-540-63531-9, pp. 94 – 110.

Meszaros, G. 1996. Patterns for Decision Making in Architectural Design. In Addendum to the proceedings of OOPSLA ’95 *OOPS Messenger* 6(4) 132 – 137.

Meszaros, G. & Doble, J. 1998. A Pattern Language for Pattern Writing. In, Martin *et al.* 1997, 529 – 574.

Microsoft Corporation, 2003. Windows XP Visual Guidelines. Available at: http://www.microsoft.com/hwdev/windowsxp/downloads/

Mikkonen, T., 1998. Formalising Design Patterns. In *Proceedings of the 20th International Conference on Sofware Engineering*. IEEE Press, Los Alamitos, CA., USA.

Molina, P.J., Torres, I. and Pastor, O. (2003) User Interface Patterns for Object-Oriented Navigation upgrade IV, 1, February 2003. Available at: http://www.upgrade-cepis.org/issues/2003/1/upgrade-vIV-1.html

Muller, M. J., Haslwanter, J. H. and Dayton, T. (1997) Participatory Practices in the Software Lifecycle. In M. Helander, T. K. Laundauer, P. Prabhu (Eds.), *Handbook of Human-Computer Interaction, Second Edition*. Elsevier Science, BV, Amsterdam.

Mullet, K. 2002. Structuring pattern languages to facilitate design. *CHI2002 Patterns in Practice: A Workshop for UI Designers* Available at http://www.welie.com/patterns/chi2002-workshop/Mullet.pdf.

Nanard, M., Nanard, J. and Kahn, P., 1998. Pushing Reuse in Hypermedia Design: Golden Rules, Design Patterns and Constructive Templates. In *Proceedings of the Ninth ACM conference on Hypertext*, pp. 11- 20.

Nielsen, J. 1994. Heuristic Evaluation. In Nielsen, J & Mack, R. L. (Eds.) *Usability Inspection Methods* New York: John Wiley & Sons.

Nielsen, J. 1993. *Usability Engineering.* Morgan Kaufmann. San Fransisco, CA, USA.

O'Neill, E. 1998. *User-developer co-operation in software development. Building common ground and usable systems*. PhD Thesis, Queen Mary & Westfield College, University of London.

Paternò, F. 2000. *Model-Based Design and Evaluation of Interactive Applications* Berlin: Spinger-Verlag.

Pemberton, L. 2000. *The Promise of Pattern Languages for Interaction Design*. Article based on presentation at HF2000, Loughborough. Available from http://www.it.bton.ac.uk/staff/lp22/HF2000.html

PLoP, 2000. Proceedings of PLoP 2000. Technical Report, WUCS-00-29, Department of Computer Science, Washington University, WA. USA. Available from http://jerry.cs.uiuc.edu/~plop/plop2k/proceedings/proceedings.html

PLoP, 2001. On-line Proceedings of the 8th Conference on Pattern Languages of Programs. Available from http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/accepted-papers.html

PLoP, 2002. On-line Proceedings of the 9th Conference on Pattern Languages of Programs. Available at: http://jerry.cs.uiuc.edu/~plop/plop2002/proceedings.html

PLoP, 1998. Proceedings of Pattern Languages of Programs ’98. *Technical Report, WUCS-98-25*. Department of Computer Science, Washington University, WA. USA. Available from: http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/

PLoP 1999, On-line Proceedings of Pattern Languages of Programs '99. Available from http://jerry.cs.uiuc.edu/~plop/plop99/proceedings/

Rheinfrank, J. & Evenson, S., 1996. Design Languages. In, Winograd, T. (Ed.) *Bringing Design to Software*. ACM Press. ISBN 0-201-85491-0. pp. 63 – 80.

Richardson, C. *A Pattern Language for J2EE Web Component Development*. Presented at PLoP 2001. Available at: http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/PLoP2001/crichardson0/PLoP2001_crichardson0_3.pdf

Riehle, D. & Zullighoven, H. 1995. A Pattern Language for Tool Construction and Integration based on the Tools & Materials Metaphor. In Coplien & Schmidt, 1995, 9 – 42.

Rossi, G., Schwabe, D. & Garrido, A., 1997. Design Reuse in Hypermedia Applications Development. In, *Proceedings of the eighth ACM conference on Hypertext.* ACM Press, ISBN: 0-89791-866-5, pp. 57 – 66.

Rossi, G., Lyardet, F. D. & Schwabe, D., 1999. Developing Hypermedia Applications with Methods and Patterns. *ACM Computing Surveys*, 31(4es) Electronic Symposium on Hypertext and Hypermedia, December 1999.

Rossi, G., Schwabe, D. & Lyardet, F., 2000. User Interface Patterns for Hypermedia Applications. In, *Proceedings of the Working Conference on Advanced Visual Interfaces*. ACM Press, ISBN: 1-58113-252-2, 136 – 142.

Roth, J. 2002. Patterns of Mobile Interaction *Personal and Ubiquitous Computing* **6** (4) 282-289.

Sandu, D. 2001. *Collection Patterns.* Presented at PLoP 2001. Available from: http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/PLoP2001/dsandu0/PLoP2001_dsandu0_1.pdf

Schmidt, D. C., 1995. Using design patterns to develop reusable object-oriented communications software. *Communications of the ACM* 38(10) 65 – 74.

Schmidt, D. C., 1996. Using Design Patterns to Guide the Development of Reusable Object-Oriented Software. *ACM Computing Surveys* **28**(4es), December 1996.

Schmidt, D. C., Fayad, M. & Johnson, R.E., 1996. Editorial. *Communications of the ACM* 39 (10). Special issue on Software Patterns. pp. 36 – 39.

Schuler, D., 2002. A Pattern Language for Living Communication. In Binder, T., Gregory, J. & Wagner, I. (Eds.) *Proceedings of the Participatory Design Conference 2002*. CPSR Press. ISBN 0-9667818-2-1. pp. 51 – 62. See also: http://www.scn.org/sphere/patterns/

Schuler, D. & Namioka, A. (Eds.), 1993. *Participatory Design: principles and practice*. Hillsdale, New Jersey, NJ. USA.

Seffah, A. 2003. Learning the Ropes: Human-Centred Design Skills and Patterns for Software Engineers' Education. *Interactions, **X**(5)*

Sharp, H, Manns, M. L., & Eckstein, J., 2003. *Evolving Pedagogical Patterns: The Work of the Pedagogical Patterns Project*. Computer Science Education, 13 (4), pp. 315-330

Smith, S. L. & Mosier, J. N., 1986. Guidelines for designing user interface software. Mitre Corporation Report MTR 9240, Mitre Corporation.

Souza, J., Matwin, S. & Japkowicz, N., 2002. *Evaluating Data Mining Models: A Pattern Language*. Presented at PLoP 2002. Available at: http://jerry.cs.uiuc.edu/~plop/plop2002/final/PLoP2002_jtsouza0_1.pdf

Sutcliffe, A. & Carroll, J. M. 1999. Designing claims for reuse in interactive systems design. *International Journal of Human-Computer Studies* **50**(3) 213 – 241.

Sutcliffe, A., 2000. On the Effective Use and Reuse of HCI Knowledge. *ACM Transactions on Computer-Human Interaction* 7(2). 197 – 221.

Tahara , Y., Ohsuga , A. and Honiden, S., 1999. Secure and efficient mobile agent application reuse using patterns. *Proceedings of the 21st International Conference on Software Engineering* May 1999. IEEE Computer Society Press. 356 - 367

Tahara , Y., Toshiba , N., Ohsuga , A. and Honiden, S., 2001. Agent system development method based on agent patterns. ACM SIGSOFT *Software Engineering Notes* 26 (3), 78 – 85

Thimbleby, H. 1990. *User Interface Design*. ACM Press, New York, NY., USA

Thomas, J. 2003. A socio-technical pattern language proposal. Available from: http://www.truthtable.com/A_Sociotechnical_Pattern_Language.html

Tidwell, J., 1998. *Interaction Patterns.* Presented at PLoP 1998. Available from: http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/P29.pdf

Tidwell, J., 1999. Common Ground A Pattern Language for Human-Computer Interface Design. Available from: http://www.mit.edu/~jtidwell/interaction_patterns.html

Tidwell, J., 1999. The Gang of Four are Guilty. Available from: http://www.mit.edu/~jtidwell/gof_are_guilty.html

Tidwell, J. 2003. UI Patterns and Techniques. Available at http://time-tripper.com/uipatterns/index.php

Towell, D. 1998. *Display Maintenance*. Presented at PLoP 1998. Available from: http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/P15.pdf

Van Duyne, D.K. Landay, J. A., Hong, J. I., 2003. The Design of Sites. Addison Wesley, Boston MA.

van Welie, M. 2002. *Interaction Design Patterns*. Available from: http://www.welie.com/patterns/index.html

van Welie, M., van der Veer, G.C., Eliëns, A., 2000. Patterns as Tools for User Interface Design. In, Farenc, Ch. & Vanderdonckt, J. (Eds.) *Tools for Working with Guidelines*, Springer-Verlag, London, 313-324.

van Welie, M., Mullet, M. & McInerney, M., 2002. Patterns in practice: a workshop for UI designers. In, *CHI '02 extended abstracts on Human factors in computer systems*. ACM Press ISBN: 1-58113-454-1, 908 – 909.

van Welie, M. 2003. *Pattern Languages in Interaction Design: Structure and Organization*: M. van Welie, G.C. van der Veer, *In*: Proceedings of Interact '03, 1-5 September, Zürich, Switserland, Eds: Rauterberg, Menozzi, Wesson, p527-534, ISBN 1-58603-363-8, IOS Press, Amsterdam, The Netherlands

Vlissides, J. M., Coplien, J. O. & Kerth, N. L., (Eds.) 1996. *Pattern Languages of Program Design 2*. Addison Wesley, Reading MA. USA etc. ISBN 0-201-60734-4.

Wake W. C., Wake, B. D. & Fox, E. A., 1996. Improving Responsiveness in Interactive Applications Using Queues. In Vlissides et al, 1996, 563 –573.

Walldius, Å, 2001. *Patterns of Recollection: The Documentary Meets Digital Media*. Aura Förlag, Stockholm.

Weiss, M., 2001. *Patterns for e-Commerce Agent Architectures: Using Agents as Delegates*. Paper presented at PLoP 2001, available from: http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/PLoP2001/mweiss0/PLoP2001_mweiss0_2.pdf

Charles Weir, James Noble. In the *Proceedings of the Eighth European Conference on Pattern Languages of Program Design* (EuroPLoP). Irsee. Universitäts Verlag Konstanz. 2003.

Windsor, P. 2000. A Project Pattern Language for User Interface Design. Presentation at *BCS HCI Group/IFIP WG 13.2 Workshop on HCI Patterns*, London, UK, November 2000.

Winn, T & Calder, P., 2002. Is this a Pattern? *IEEE Software*, January/February 2002, pp.59-65.

Wirfs-Brock, A., Vlissedes, J., Cunningham, W., Johnson, R. & Bollette, L., 1991. Designing Reusable Designs (panel session): Experiences Designing Object Oriented

Frameworks. In *Proceedings of OOPSLA / ECOOP 90,* Addendum: systems, languages, and applications ACM Press, New York, pp. 19 - 24.

Wynn, E. and Novick, D.G., 1995. Conversational Conventions and Participation in Cross-Functional Design Teams. In Proceedings of COOCS, 95, ACM Press, pp. 250 – 257.

Yacoub, S. M. & Ammar, H. H. 1998. A Pattern Language of Statecharts. Presented at PLoP 1998. Available from: http://citeseer.nj.nec.com/yacoub98pattern.html